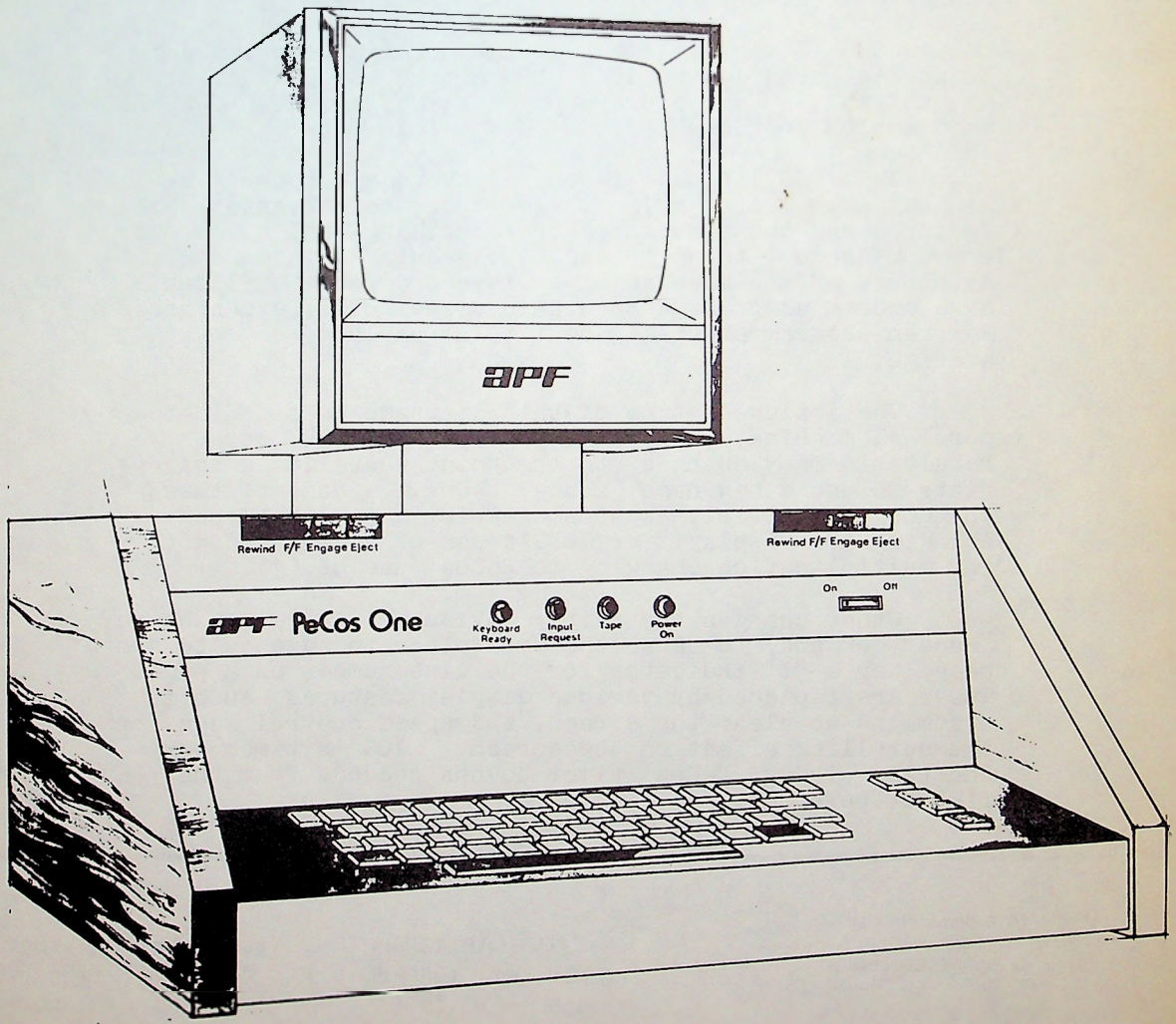
$2.50

65

Popular Computing

The world's only magazine devoted to the art of computing.

August 1978 Volume 6 Number 8



PeCos One A New Concept in Personal Computers

A forthcoming entry in the personal computing field is PeCos One (Personal Computing System) produced by APF Electronics. The desk-top unit has a 60-key keyboard, a CRT display, and two built-in cassette drives. Its microprocessor is the 6502. Its unique feature, however, is its built-in interpretive language, also called PeCos, which is derived from the JOSS language of the RAND Corporation.

JOSS began in 1964, the creation of J. C. Shaw, originally implemented on the JOHNNIAC computer. It was one of the first truly conversational languages, and originally operated in time-sharing mode in the RAND building in Santa Monica, using eight old IBM typewriters for consoles. The JOSS reference manual was condensed to a one-page summary in 1965 and has endured with little change ever since.

JOSS II was implemented with a PDP-6 computer and 32 special consoles built by DEC around IBM Selectric mechanisms. User storage on disks was added. JOSS III used an IBM 360 computer.

The JOSS language is one of the easiest computer languages to learn, primarily because the conversational features and the elaborate error checking--and courteous error messages--tend to lead people quickly into sophisticated use of the machine. Novices and young children have become adept at JOSS within a few hours; experienced Fortran programmers take about twice as long, on the average.

The implementation of this language on a dedicated personal machine, with primary output to a CRT screen, results in many changes, to the point where it is appropriate to use a new name; hence, PeCos. Some of these changes are trivial, as, for example, the verb "Type" changing to "Display," or JOSS's use of a centered dot for multiplication changing to PeCos' use of (*).

Other changes are fairly obvious. There can be no "Page" command, to cause the typewriter to advance to a new page, nor a "$" indicator for the line number on a page. These are replaced by various display features, such as a command to clear the screen, and speed control keys for the scrolling effect on the screen. JOSS's timer gave the time of day; PeCos' timer counts seconds from the time of power on.

Publisher: Audrey Gruenberger
Editor: Fred Gruenberger
Associate Editors: David Babcock
Irwin Greenwald
Contributing Editors: Richard Andree
William C. McGee
Thomas R. Parkin
Advertising Manager: Ken W. Sim
Art Director: John G. Scott
Business Manager: Ben Moore

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each.

Other changes between the two versions are more significant. The Let verb of JOSS is not implemented in the first production version of PeCos. PeCos is not as finicky as was JOSS about magic values; for example, in PeCos, $\sin^2 X + \cos^2 X$ may be .999999996 (X = 2.75456789) whereas JOSS would give 1.0 for most valid values of X. PeCos clearly has no digits to spare in its arithmetic (and one can sympathize with efforts to program nine significant decimal digit arithmetic in an 8-bit machine language), so that we have things like:

exp(12) = 162754.791 (JOSS)

exp(12) = 162754.794 (PeCos)

exp(12) = 162754.791419 (True).

To illustrate conversational computing in action, and the PeCos language in particular, the details of solving one of our old familiar problems are given here.

> The 2-3-5 problem was first presented in issue No. 10. Three different solutions were given in issue No. 16, including one by the problem's author, Richard Hamming. Finally, in issue No. 47, another solution, by Norman Sanders, was given.
>
> The problem is simple: list, in sequence, all integers whose factors consist only of 2, 3, and/or 5. This list begins: 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20,...

Following is a copy of the PeCos display as a program is developed, together with explanatory notes. This material is paraphrased from the corresponding JOSS program that appeared in the book Computing: A Second Course (Canfield Press, 1971).

The real value and power of conversational computing can be felt only by sitting at a console and using it, preferably on a new problem. However, by following the gropings and blind alleys of a novice programmer on a simple problem, it is possible to get some of the feel of this form of the computing art.

```
6.1  Set N = 1.                                    (A)
6.2  Set N = N + 1.
6.3  Set X = N.                                    (B)
6.4  To step 6.6 if fp(X/2) ≠ Ø.                   (C)
6.5  To step 6.8 if fp(X/3) ≠ Ø.
6.85 To step 6.9 if fp(X/5) ≠ Ø.
6.86 To step 6.2.
6.6  Set X = ip(X/2).                              (D)
6.63 To step 6.95 if X = 1.
6.64 To step 6.4.
6.8  To step 6.95 if   CLR                         (E)
6.8  Set X = ip(X/3).
6.83 To step 6.95 if X = 1.
6.84 To step 6.5.
6.9  Set X = ip(X/5).
6.91 To step 6.95 if X = 1.
6.92 To step 6.85.
6.95 Display N in form 5.                          (F)
Form 5:

-----

Do part 6.                                         (G)
Interrupted at step 6.6.
Display all values.
Not yet available.                                 (H)
Display N, X.
     N = 2
     X = Ø
```

A program developed in the PeCos language. The circled letters refer to the notes, beginning on page 7.

6.4 To step 6.6 if fp(X/2) = ø.

6.5 To step 6.8 if fp(X/3) = ø.

6.51 To step 6.85 if fp (X/5) = ø.

Do part 6.

2

Display size. (J)

Size = 1779

6.96 To step 6.2.

Do part 6.

2

I'm at step 6.51.

Eh? (K)

6.51 To step 6.85 if fp(X/5) = ø.

Do part 6.

2
3
4
6
7
8
9
12
13
14

Interrupted at step 6.4 (L)

Deleete step CLR (M)

Delete step 6.85.

6.51 To step 6.9 if fp(X/5) = ø.

Do part 6.

2
3
4
5
6
7
8
9
10

6.52 To step 6.2.

Do part 6.

2
3
4
5
6
8
9
10
12
15
16
18
20
24

I'm at step 6.92.

I can't find step 6.85.

6.92 To step 6.51.

Do part 6.

[Correct results now.]

6.15 Set k = Ø.

6.95 Set k = k + 1.

6.96 Display N, k, timer in form 5.

6.97 To step 6.2.

Form 5:

---- ---- ----

[With some results after restart:]

N	k	timer
2	1	4646
1ØØ	33	4729
5ØØ	66	5Ø16
1ØØØ	85	5374
16ØØ	1ØØ	58Ø4
1728	1Ø2	5895
3125	124	6895
5ØØØ	143	8246

Notes

A. The solution to the problem begins with a scheme to generate all the natural numbers, starting with one. The step number (6.1) is arbitrary. In the terminology of PeCos, the 6 is called a "part," and steps with a part are numbered 6.1, 6.2, and so on. If a step needs to be inserted between steps 6.1 and 6.2, it can be numbered 6.13, Dewey-decimal fashion. As each numbered step is entered via the keyboard, it is stored, to be executed later. No analysis of a statement that is to be executed later is made when it is entered.

B. For each natural number that is generated, a series of tests will be devised to determine whether or not it should be printed or displayed. Since the decisions will require operating on, and hence destroying, the number being tested, it is copied by step 6.3. The value of X can then be manipulated, and the original value N will be retained in storage.

C. The decision logic begins here. Control is to pass to step 6.6 (that is, not to advance to the step whose number follows in sequence after 6.4) if the fraction part of X/2 is not zero. Step 6.4 asks the logical question "Does 2 divide X?" The initial implementation of PeCos belongs to the zero-slashing school.

D. We arrive at step 6.6 when 2 divides X. Step 6.6 carries out the division, and replaces the value of X by X/2. Similarly, step 6.8 replaces X by X/3, and step 6.9 replaces X by X/5.

E. A faulty line is erased by means of the CLR key. A stored statement that begins or ends with an asterisk is taken to be a COMMENT.

F. Step 6.95 calls for output during execution of the program. "Form 5" is defined just below, and specifies that the value of N is to be displayed as an integer, with a maximum of four digits.

G. We now have a small program in storage, but it is not debugged or tested. The direct command (meaning "do it now") at G calls for execution, in order, of all steps whose step numbers begin with 6. In this case, nothing happens except that the "Keyboard Ready" light goes out, indicating that computation is taking place, but with no output, not even an error message. Thus, the program is partially debugged, in the sense that it does execute and that PeCos can detect no logical or syntactical errors.

H. An examination of the current values of all variables in the problem would help to determine the bug. The command "Display all values" is from JOSS, to be added eventually to PeCos. The fact that N is 2 and X is zero helps to reveal that the logic of the tests is backward, so the three statements for the tests are changed by re-keying, which replaces them in storage. The program is again triggered.

J. The program outputs a 2 and then hangs up. For no good reason, the machine is interrogated to find out the amount of storage still unused: 1779 units. At power-on time, the user of PeCos has 1864 units of storage. A single variable will consume one unit. A statement like 4.3 Set X = 5 will consume 4 units. The statement:

5.5 Set Y = sqrt(B*B - 4*A*C).

consumes 7 units.

The command "Display part 6" will bring all the program to the screen for examination. It is readily seen that the program has a dangling end; after step 6.95 there is nowhere to go. Step 6.96 is the patch, and the program is restarted.

K. "Eh?" is the all-purpose error message for simple errors in making up statements. The version of step 6.51 that was entered contained an illegal space: fp (X/5). The correct version of step 6.51 is entered, and the program is again triggered.

L. The program now outputs many numbers, but the list is in error; it should not contain numbers like 7, 13, or 14.

M. Another use of the CLR key--there is no point to continuing a statement with a misspelled word. It is easier to erase the entire statement than to backspace to the point of error. A correction is made, and yet another attempt at execution shows new trouble: we are getting all integers now.

N. One more correction, and the program begins to produce correct output. But after the output of 14 numbers, it is revealed that one of the "corrections" has caused trouble, which is easily fixed. The program now works and seems to produce correct output.

P. Under the maxim "First make it work, then make it pretty," we can now dress up the output to count the numbers as they are displayed and, to show the speed of the machine, the value of the timer. The present version of PeCos takes just one hour to calculate 143 numbers, during which time 5000 natural numbers were examined.

In issue 46 we reported an advertisement by CBM Commodore, U.K., Ltd., which suggested that you test any "scientific" calculator by carrying out the sequence:

sin cos tan sqrt log exp square $\tan^{-1}$ $\cos^{-1}$ $\sin^{-1}$.

They suggested using the value 29 (degrees), for which the sequence of operations yields:

Commodore's SR4190R: 29.00100887

TI SR-52: 29.00001537

various Hewlett-Packard machines: 29.00xxxx

PeCos doesn't have arctangent as such, but rather the more general function arg(x,y), which is the arctangent of the point (x,y). In addition, we have that

$$\arcsin(x) = \arg(\sqrt{1 - x^2}, x)$$

$$\arccos(x) = \arg(x, \sqrt{1 - x^2})$$

So, to give PeCos' arithmetic and function subroutines a good workout, we can use the Commodore test and execute this program:

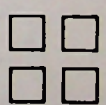
```
4.11  Set a = sin(x).
4.12  Set b = cos(a).
4.13  Set c = sin(b)/cos(b).
4.14  Set d = sqrt(c).
4.15  Set e = log(d).
4.16  Set f = exp(e).
4.17  Set g = f↑2.
4.18  Set h = arg(1,g).
4.19  Set j = arg(h,sqrt(1 - h↑2)).
4.20  Set k = arg(sqrt(1 - j↑2),j).
4.21  Display x, k, |x - k| , _.

Do part 4 for x = .01(.01)1.00.
```

The running of this program shows that x and k will differ by values of the order of .00000004. In particular, for x = .506145483 (that is, 29 degrees, expressed in radians), PeCos furnishes .506145549, which is equal to 29.00000378 degrees. Hence we can conclude that, in general, the arithmetic subroutines in PeCos are well written.

Fourteen years of use of the JOSS language have certified it as a powerful computational tool because of the intimate interaction it fosters between man and machine. With such a tool now available, via PeCos, on a personal computer, the need for professional programmers should attenuate, at least for a broad class of problems.

The programming displayed in the example above is rather crude, but it has been deliberately left that way to illustrate how non-professionals can be led by a clever language to a problem solution in a way that is distinctly different from the usual high-level language approach.

The PeCos One machine is a break-through from the tyranny of BASIC interpreters and the inefficient programs that usually result from the use of that language.



PC65-10

Where Does Personal Computing Stand?

by Fred Gruenberger, Editor, POPULAR COMPUTING

The revolution (a fair term, I think: "A total or radical change") that we now denote by *personal computing* began early in 1975. As things go in computing — our whole field is only a little over three decades old — we have thus gone far enough to justify an overview. There have already been three articles tracing the *history* of personal computing.

There are now eleven journals in this sub-field, all but three of them monthlies. There couldn't possibly be enough new material to fill eleven journals, even if they printed every neophyte's programs intact (which they appear to do), so they are beginning to hurt for material, and it shows. Several of them have descended from ham radio sources, and are most happy when they deal in circuits and describe new adventures in soldering. Some infighting has broken out, and the competition for the advertisers' dollars is keen. I would venture to predict that two years from now at least half of the current journals will have disappeared.

One big problem that the magazines have is that they can't seem to define their audience, nor focus on any one part of it. It is clear that there are junior high school students making effective use of personal machines, but there are also graduate engineers using them and — what has been quite surprising — professional computer people, too. Some of these people are interested in file processing; some are interested only in game playing; a small group has taken an interest in numerical methods; there is surely a group that has invested in a machine by mistake ("Send me the program to do ESP on my PET"); and a large group who don't want to run their machine at all — they apparently want only to look at it. It should be evident that no journal can possibly appeal to all these groups at once. Most of the magazines have tried to and, just to cover all cases, have added science fiction stories and how-to-build articles on bug-eyed monsters. One longs for a magazine with the clear cut mission of *The Bee-Keeper's Journal*. Cuteness (childish names, comic strips, and ninth grade vocabulary) may attract the junior high school crowd for a while, but sooner or later, even that group will grow up and resent being patronized. The magazines that survive will be the ones that decide on an audience and a level of education and then serve that group.

There is much of value to be found in the magazines, of course. With a small amount of reading, the rank beginner can easily avoid costly mistakes. The trouble is, the gold is mixed with such a lot of dross. Nearly every journal can't resist publishing every complimentary letter they get, even the illiterate ones. At the moment, each journal feels obliged to present an evaluation of the Heath, Commodore, and Radio Shack machines. In nearly every case, it seems that the reviewer has not used any other machine, and is evaluating the machine he bought in comparison to his pocket calculator.

The level of writing for many articles is pathetic, both in terms of computing and English. Total misconceptions are rampant, as evidenced by the use of words like MNEUMONICS and KLUGE (the correct version and meaning of *mnemonic* could be found in any dictionary; *kludge* is an in-term, and if you don't know about it, you shouldn't use it). Authors seem positively proud of their ignorance and naivete: they have discovered that factorials grow beyond their system's range quite rapidly; that permutations aren't easy to come by; and that the square of *a plus b* is the square of *a* plus the square of *b* plus twice the product of *a* and *b for all values of a and b*. Each month some adventurer proudly announces a discovery (e.g., listening in on the computer via an FM radio) that is as old as computing. On the average, these monthly discoveries are now up to 1954. It is understandable why the magazines are flooded with ancient material newly rediscovered — the enthusiasm of the newcomers is one of the big pluses of the revolution — but it is dismaying that the stuff is printed. The sad fact is that the editors are not computer people; they don't even know that they're publishing misinformation, or old, old stuff, or half-baked programs. On the other hand, when massive computing power is made available to masses of people at low cost, then there are bound to be significant effects: new applications, novel uses, interesting methods of attack on old problems — all the things that fresh eyes can give to an old subject. Despite what we have seen so far, big things are sure to emerge from the impact of personal computers.

The hardware that is already available is quite good; that is, it is reliable, low priced, human engineered, with adequate provision for servicing. There have been few instances of consumer rip-offs in the hardware end of the business. The software is not so good, and little is known about how bad it is. Nearly all the journals publish enormous complete programs, always in BASIC, and almost always at or near the threshold of readability. Many of these programs are awe-inspiring examples of bad coding (and invariably they appear in the same issue with an article telling of the wonders of structured programming, and how it makes master programmers out of muddled thinkers). It doesn't seem to occur to either the writers or the editors to offer the *algorithm* involved, rather than the program. If a novel application were presented through a flowchart or other tool of communication, then the reader a) would have the fun of writing his own program, and b) could readily re-code it some day when BASIC disappears. Incidentally, I have yet to see a program presented that shows its test procedure and test results.

We seem to be approaching the level of one personal computing convention a week — surely this is really massive overkill — and some of these must be rip-offs. The proliferation of "conventions" will probably become self regulating; the exhibitors at these shows, who pay the bills, will soon sift the good ones from the phony ones.

Up to now, every personal machine has offered a high level language, if at all, via an interpreter; some of them offer an assembler. So far, every machine has had BASIC; one machine offers APL. The pending PeCos machine will have JOSS, which will also be interpretive. As the shift to 16-bit microprocessors takes effect, and central storage becomes cheaper, it is reasonable to expect that we may see real compilers on personal machines; the first of these will be, of course, a BASIC compiler.

The makers of the machines must be leading a hectic life. Competition is alive and healthy, but so is technical progress, and any vendor who says — even for an hour — "We've got it made," won't have, a day or so later. The 16-bit CPU's are just now being fabricated; when they get past their shakedown phase, say in a year, the 8-bit processors will vanish immediately.

All of the above adds up to one thing: furious activity, dynamic change, a lucrative market, and growth that is exponential and shows no signs of fading. Clearly, people will buy almost anything (eleven journals, for example) and most manufacturers can't keep up with their orders. Whichever way things go, it promises to be exciting.

Reprinted by permission from Data Processing Digest, Vol. 24, No. 6, June 1978

R_{317} Is Prime

An area of particular importance in number theory, the branch of mathematics concerned with the properties of integers, is the analysis of prime numbers: any number that can be exactly divided only by itself and 1. Although primes have been studied since the time of Euclid, there is still no efficient way to recognize and prove that a large number is prime. Of course, dividing the number by all smaller prime numbers will show whether or not it is prime, but even with today's fast computers that procedure is prohibitively time-consuming for large numbers. Therefore it is of special interest that Hugh C. Williams of the University of Manitoba has discovered a new prime number. The number is 11,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111,111,111,111,111,111,111,111,111,-111. It is called R_{317} for short.

The new prime is a member of an elite group of numbers; it is a rep-unit number, that is, all its digits are 1. Each rep-unit number, R_n, is equal to $10^n - 1$ divided by 9, and so n is the number of 1's in R_n. For example, R_1 is equal to 1, R_2 is equal to 11 and so on. It is easy to see that if R_n is prime, then n is equal to a prime number, p. (If a divides n, then R_a will divide R_n exactly.) R_p is not, however, necessarily prime. In fact, until Williams discovered R_{317} last year there were only three known rep-unit primes: R_2, R_{19} and R_{23}. Almost 50 years had elapsed since the last discovery of a rep-unit prime.

Williams did not set out to find the fourth rep-unit prime but gave a computer problem to Eric Seah, an undergraduate student. Seah was to run a pseudoprimality test on all the R_p's for which p is less than 1,000. The test, based on the work of the 17th-century mathematician Pierre de Fermat, consists in looking at the expression $13^{R_p - 1}$ to see whether it is equal to 1 modulo R_p, that is, whether the remainder is 1 when $13^{R_p - 1}$ is divided by R_p. The procedure is called a pseudoprimality test because if an integer fails the test (if the remainder is not 1), then the integer is definitely not prime, but if the integer passes the test (if the remainder is 1), then the integer is probably prime.

It seemed that nothing new could be expected from Seah's testing of the R_p's because they had all been tested before. John D. Brillhart of the University of Arizona and John L. Selfridge of Northern Illinois University had tested all the rep-unit numbers from R_1 to R_{358} and had determined that all except the three known primes were composite numbers. Other mathematicians had extended the list past $R_{1,000}$.

Seah began with a preliminary sift of the R_p's, dividing each one by every prime smaller than 1,000 and discarding those R_p's that divided exactly. Next he ran the pseudoprimality test on the remaining R_p's. Surprisingly, four rep-unit numbers passed the test: R_2, R_{19}, R_{23} and R_{317}. Seah's results convinced Williams that R_{317} was indeed prime. "If a number passes one pseudoprimality test," he said, "in your heart you know it is prime." Even so, Williams retested R_{317} for pseudoprimality with three different computer programs and 50 variations of the pseudoprimality test. Each time R_{317} turned out to be a pseudoprime. "The real problem," Williams said, "was to prove that it was prime."

It is extremely difficult to prove that such a large number is prime. Williams chose to prove that R_{317} is prime by finding several factors for the number $R_{317} - 1$. The discovery of each factor of $R_{317} - 1$ places strong restrictions on the possible factors of R_{317}. Therefore if R_{317} is prime, when a sufficient number of factors have been found for $R_{317} - 1$, no possible factors remain for R_{317} (except, of course, itself and 1). Williams was unable to factor $R_{317} - 1$ sufficiently, and so he sent the problem to Brillhart and also to D. H. Lehmer of the University of California at Berkeley, who had access to the powerful computer ILLIAC IV. Brillhart responded first, saying that Samuel S. Wagstaff of the University of Illinois had discovered a large divisor of $10^{79} + 1$, one of Williams' remaining composite factors. Two weeks later Williams received a letter from Lehmer, who had computed the same divisor. With this new information Williams completed the proof that R_{317} is prime.

Williams asserts that his discovery of R_{317} involved "a lot of luck." Primality testing is difficult because at present all primality-testing algorithms involve factoring numbers, which is very hard to do in a reasonable amount of time. On the other hand, recent developments in the techniques of factorization and primality testing are bringing larger numbers within the scope of modern computers. The discovery of R_{317} provides no information about the existence of other rep-unit primes, but there is no reason to believe there are not more of them. The mistake in the work of Brillhart and Selfridge was probably the result of human error in typing rep-unit numbers, but assuming that the rest of the pseudoprimality testing that has been done is valid, the next candidate for a rep-unit prime is $R_{1,031}$.

repunit numbers

The article on page 11 is reprinted, with permission, from the February 1978 issue of Scientific American. The article refers indirectly to the paper by Brillhart and Selfridge,

"Some Factorizations of $2^n \pm 1$ and Related Results"

in Mathematics of Computation, V. 21, January 1967. The pertinent section reads:

> "... the numbers $N_p = (10^p - 1)/9$, p prime, are shown to be prime for p = 2, 19, and 23, and for no other $p \leq 109$. We have extended the search for primes of this form, but there are no further primes for $p \leq 359$. This result was obtained by showing that
>
> $$a^{N_p-1} \not\equiv 1 \mod N_p$$
>
> for those N_p for which no factor was known. For each N_p this test was run twice for a = 3 with complete agreement in the final remainders."

The fact that an error was made in a small corner of an esoteric discipline is not "a pivotal event in world history." The way the error came about is interesting, however, as is the fact that the world believed the erroneous result for 15 years because of the reputation of the two men involved.

Prof. Hugh Williams wrote as follows:

"With regard to your question concerning the number R_{317}, I could not agree with you more about Selfridge's care in writing papers. In fact, when I first discovered the possible primality of R_{317} as long ago as 1974, I did not believe that my result was correct because it contradicted the results of Brillhart and Selfridge in their paper. However, when I came to write my current paper, "Primes with Interesting Digit Patterns," I thought that this number should be re-examined, so I asked Eric Seah to write a program to retest the numbers R_n for $n \geq 23$. Seah's results seemed to confirm my earlier suspicions, which, by the way, came about as a result of work by my student Stephen Judd. Consequently, we decided to work on the much more difficult problem of proving primality. I should emphasize that this is a much more difficult problem. It is very easy to identify these numbers as probably primes, but it is extremely difficult to prove primality in a mathematically acceptable manner. With R_{317} we were extremely lucky. I wrote John Brillhart about [the discrepancy] and he returned the following reply to me:

"I was very interested in your results on N_{317}. It is highly unlikely that the program I used to test N_{317} and the other numbers was wrong. I used the standard routine I had been using for years and it was well checked out. Also, since I tested each number twice and on different days and compared the remainders (the tests were a couple of days apart, as I recall) it seems unlikely that it was a machine error, unless the 7094 was consistently misbehaving then.

"If there is an error here (which there might be), it would probably have come from the fact that each number to be tested was punched in decimal, 80 digits per card, and then read in through the card reader. Thus, N_{317} would have been 317 ones, punched on four cards, the first three having 79, 80, 80 columns all ones (the first column of the first card was for the sign) and the last having 78 ones.

"Since the input was printed as well as the output, I'm sure I would have noted if the card reader had not input all ones. Thus, the only possibility I can think of was that I miscounted the number of ones, possibly only punching ones up to column 77 or 79 on the last card.

"Unfortunately I no longer have either the cards or the output to check, so I guess I'll have to chalk it up as an operator error. Sorry to cause you any problems about that. I've always tried to be very careful, but these things really need to be checked!"

And Prof. Williams goes on to say:

"As you can see, the problem here was that the authors were not using the computer to generate the numbers they were testing in core. This is always a mistake; as you well know, one should let the computer do everything. I should point out here that the error which they made was a very small one and affected only one sentence in their entire paper, and there is no reason to believe that the entire work has been vitiated."

The Scientific American article concludes that attention now shifts to R_{1031}. With an eye toward making a start at finding the fifth prime repunit number, Associate Editor David Babcock made a preliminary run on his Apple II computer. For each of the repunit numbers from R_{1031} through R_{1489} for which the index number is prime (that is, the ones that are eligible), the program formed the remainder on division by the primes up to 3271 (an arbitrary cutoff point)--the "preliminary sift" mentioned in the article. This computer run produced these results:

R_{1439} is divisible by 2879

R_{1481} is divisible by 2963.

This is not new information. Samuel Yates has reported (in the Journal of Recreational Mathematics) on known factors of 28 of the 65 numbers tested by Babcock's program, including, of course, the above results. What is of interest here is the timing information on the preliminary run; namely, that it took a 32 hour run. This gives us some indication of the enormous computing effort it will take to establish the next prime repunit number.

In issue 55, Problem 209 from James L. Boettler called for finding five positive distinct integers such that every pair of them sums to a perfect square.

The problem originated from an offer by A. R. Thatcher (in The Mathematical Gazette, March 1977) of £25 for the best solution to the problem. The March 1978 issue of the Gazette presents mathematical analyses (augmented by computer searches) by three different people (dating back as far as 1971), resulting in many solutions, among which are these:

7442 28 658 148 583 177 458 763 442

30 823 058 63 849 842 150 187 058
352 514 183 1 727 301 842

and a set of six numbers, one of which is negative:

-15 863 902 17 798 783 21 126 338 49 064 546
82 221 218 447 422 978

ADVERTISING IN COMPUTING:

THE STATE-OF-THE-ART IS DISMAL.

Commentary by Scott G

Most advertising of computer ware, either hard or soft, is bad beyond belief.

Not simply bad in the mechanics of typesetting, design, readability, and clarity of image, but bad in a conceptual sense. Gimmicks abound: astronauts are displayed in a feeble attempt to market storage; skeletons are used to peddle rapid report generation; real people (as we tend to refer to non-actors) are featured in unimaginative and boring testimonials. At least 50% of the ads in trade publications have a ridiculous concept or none at all.

One example of a good concept gone sour is the computer advertiser's use of the testimonial. Users of the product are induced to make some inane statement like "We couldn't live without Product X!" as though we could possibly care about their health.

There is a real lack of economical writing. So much so, that a simple statement of fact shines like pearls before swine. "Acme announces the world's lowest priced mainframe" is a welcome relief after encountering such weird concepts as "Cassidy digital tape transports and the Quality Explosion" or "It's no wonder the 9900 series is selling like hotcakes!" or "Intelligence enters the picture: Now big-screen graphics does bigger jobs in better time."

Probably the worst aspect of the industry's ads is the horrible clutter. A competent designer may have developed the original layouts, but it is obvious that a committee of anesthetic bozos reworked the ads into visual mush.

Designs utilizing blatant colors are very popular, as are designs with at least nine different type fonts. Often, these ads are typeset with a proportional spacing typewriter instead of by a professional type house. The ads are then assembled by an in-house artist whose only function up to that point was printing the firm's monthly newsletter.

Computer advertisers, please consider your image. Every time you sell your product or service, you're also selling your firm. Your corporate identity is important, both to you and to your customers.

After examining a recent issue of *Datamation*, the following words of advice are offered:

*Because corporate image is very important, it's best not to borrow fast food franchise slogans. Using Burger King's "Have it your way" is not a good idea in selling to the computing industry.

*Appropriating the "What if . . ." line from IBM is not likely to convince query language users to consider your product, particularly if your ad is a textbook example of ugly design.

*Offering a salary survey as a premium is acceptable; listing the 35 branch offices where you can get the premium is fine; doing both in what appears to be a college newspaper paste-up job is foolish.

*Phrases like "More than a computer company," like the words "turkey" and "loser," should be avoided.

*Showing your product casting long shadows in the shape of your firm's name might be effective if A) you could easily read the name with its elongated type face; and B) there was some good reason for doing so in the first place.

*Don't put your company officers into your ads. Nobody cares what they look like. If you need people to carry out the concept of your ad, hire professionals. They know how to stand, wear a suit, hold their arms, etc.

RULES TO FOLLOW IF YOU DISAGREE WITH THIS ARTICLE

1. Be certain to let the CEO's nephew design the ad campaign. There is no sense paying the high price art directors charge these days.

2. Readership increases as the square of the number of type styles.

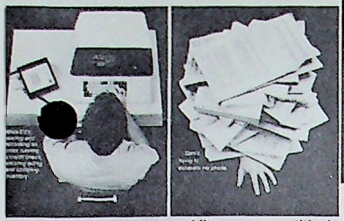
3. By all means stress facts which are irrelevant to your product. Facts of any kind are helpful in large quantities.

4. The bigger the gimmick in your ad, the better, especially if it distracts from your firm's corporate identity.

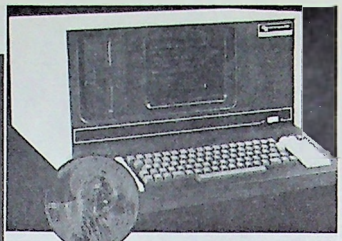
5. Always let the magazine "pub-set" your ad. This saves on the cost of typesetting, and the results will always be stunning.

Scott G, who came by his last name in a fit of anti-bureaucratic pique, is a copywriter at Webb & Silberg Advertising in Los Angeles.

With XL40 Distributed Processing, your people will be doing business...



...while your competition's doing paperwork.



MDT-400...The sensible intelligent terminal at a sensible price,

for text Editing, Distributed Processing or Communications.

MDT-400
It makes sense.

compugraphic

Give us 3 minutes

and we'll show you how to cure your 2501 headache

DOCUMATION INCORPORATED

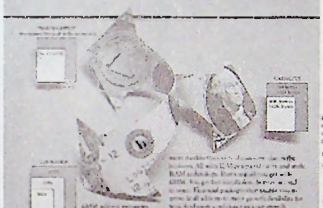
Don't let static kiss your computer's memory good-bye!

3M

"We couldn't live without MARK IV!"

informatics Inc

More ways to grow with EMM add-on memory.



The difference in add-on memory

EMM COMPUTER PRODUCTS

as/inquiry

The IMS DB DC QUERY LANGUAGE

THE TALLY PRINTER TERMINAL

Control 42 functions via the keyboard or the program

"Control Data's Mass Storage System was the better business decision for us."

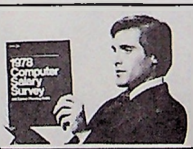
CONTROL DATA CORPORATION

More than a computer company

New! 1978 Computer Salary Survey

Call for your FREE copy today!

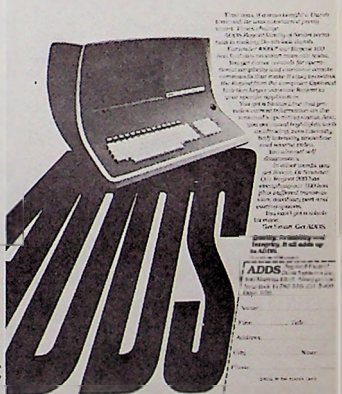
source edp

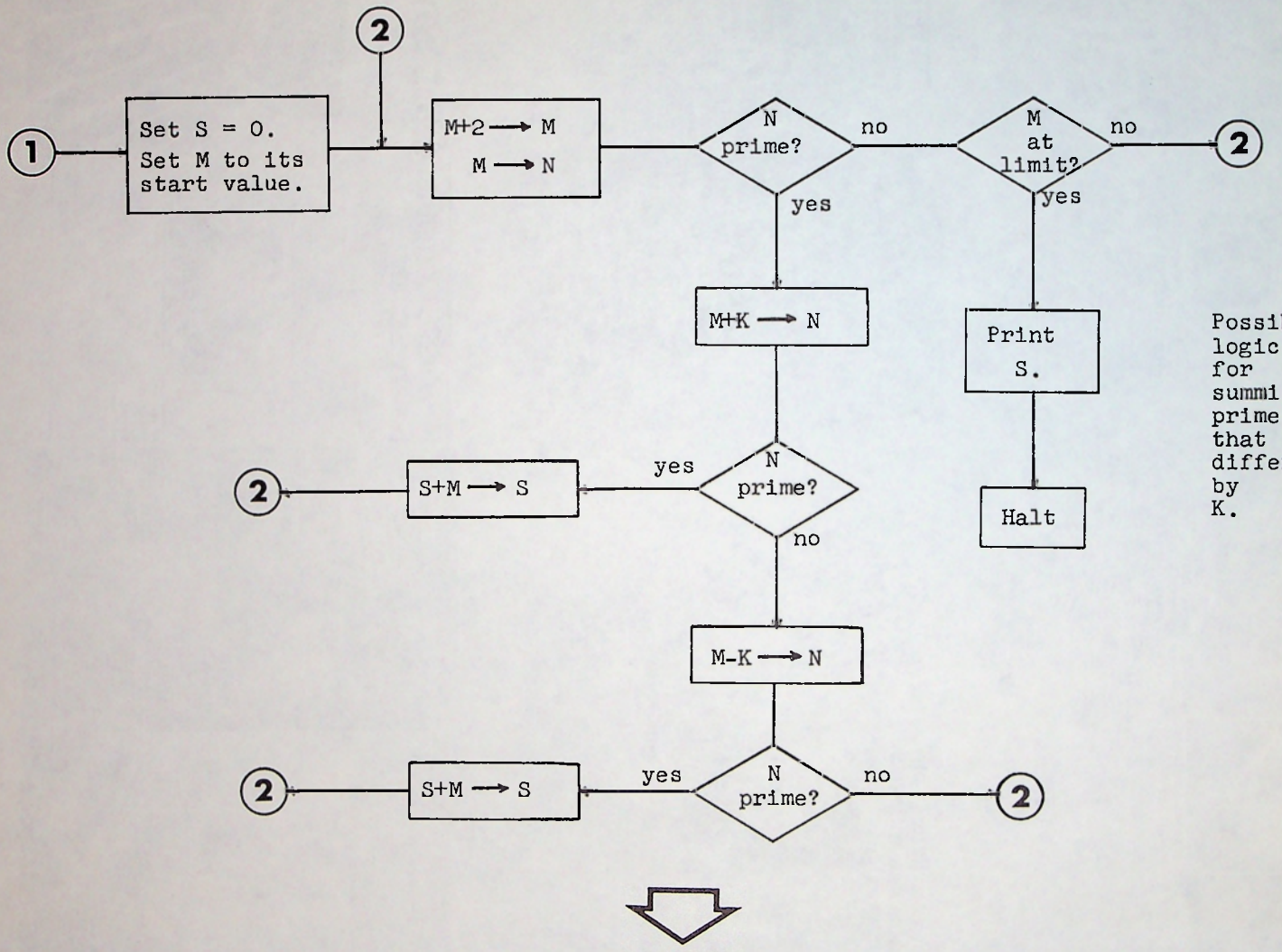


For your FREE 1978 Computer Salary Survey call the Source Edp office nearest you.

Get Smart.

ADDS Regent · Smart for the price of Dumb





Possible logic for summing primes that differ by K.

Range		4	6	8	10	12
1001	1999	70741 47	128921 87	71298 50	97969 67	128760 88
2001	2999	112751 45	170509 69	120264 48	155088 62	176587 71
3001	3999	146754 42	206991 59	132846 38	162732 46	255891 73
4001	4999	123889 27	319881 71	132834 30	234716 52	282689 63
5001	5999	215285 39	374946 68	220284 40	231928 42	362758 66
6001	6999	163681 25	377596 58	298758 46	343831 53	404024 62
7001	7999	265715 35	368729 49	208308 28	291989 39	384683 51
8001	8999	289746 34	500831 59	264641 31	356316 42	442668 52
9001	9999	283518 30	616077 65	275395 29	341244 36	569012 60

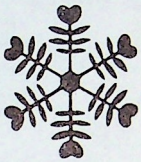
Our second Homework problem (issue No. 63, page 13) called for summing those prime numbers, p, for which p+6 or p-6 is also a prime.

The table above shows some results in various ranges, for primes differing by 4, 6, 8, 10, and 12. The smaller number with each table entry is the number of primes involved in the sum.

The accompanying flowchart shows a possible scheme for this calculation, where K is the difference value--4, 6, 8, 10, or 12.

We solicit more entries for this table, both in higher ranges and for higher difference values.

Word has been received of a table of prime numbers up to 2,000,000 in the natural numbers (which would amount to 148,910 primes), available in full size printout for $20 per copy if there is sufficient demand for it. The possible demand will be accumulated by Harvey Dubner, Dubner Computing Systems, 1740 Broadway, New York 10019.



homework 3

...this one due to Linda Hawkes

Draw a structured flowchart for the following recipe:

Carl Compass' Casserole

77777 Blank cards, shredded
777 (each) FINIS and $OBJ,LGØ cards, finely diced
49 cans Cream of Elephant soup
49 cups EØJ card chad
onions, as needed*

Simmer blank cards in enough tears* to cover, until tender. Drain and mix with soup and seasoning (blue and green cards). Place in oiled, covered casserole dish. Sprinkle red chad over the top. Bake in preheated 536B° oven until bubbling in the center. Serves 120.